Assignment 4: Reinforcement Learning

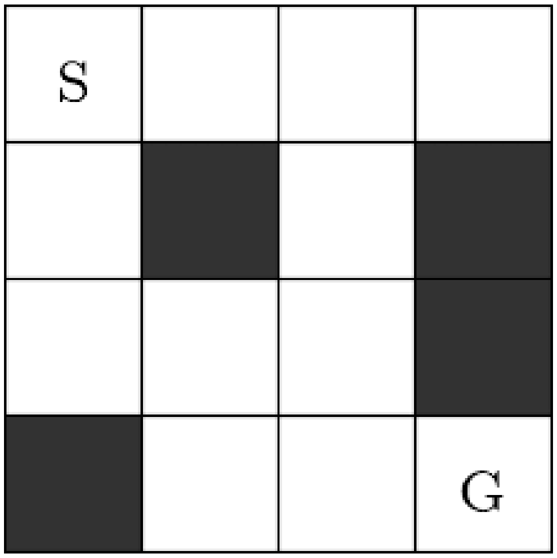
[Jainil Modi]
[jmodi30@gatech.edu]

I. Introduction

For this assignment, we will explore two problems: the Frozen Lake Problem, and the Jack's Car Rental problem. In this assignment, we are also tasked with making one of problems "small" and the other "large". I chose to make the Frozen Lake problem "small", with 16 states, and the Jack's Car Rental problem large, with 441 states. We will explore both of the problems, the three methods of solving them are policy iteration, value iteration, and q-learning. The problems and their setups will be explained in their respective sections below.

II. MDP 1: Frozen Lake Problem

In the Frozen Lake problem, we are tasked with navigating exactly that: a frozen lake. We want to avoid any holes so we do not fall in, and so we make it across. For the purposes of my setup, there are four actions the agent can take: up, down, left, right, and no move. Each have a probability of 0.1. The probability that the agent does not move is 0.8. Below is an example of how a frozen lake problem might look:



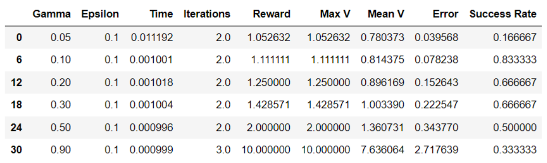
The s is the start, the black squares are holes, and g is the goal. The unlabeled white boxes are frozen and walkable.

What does it mean for this problem to be solved? The agent makes it across the frozen lake safely without falling into the ice.

A. Policy Iteration

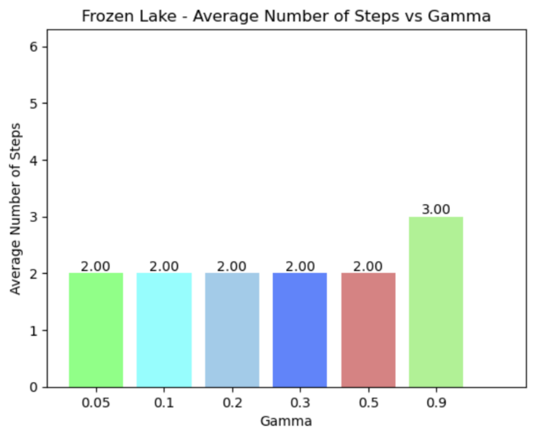
In this method, we explore how to maximize the utility of the current policy and try to improve within it. Convergence of this algorithm is defined as when the delta value δ drops below a threshold epsilon ϵ. In this portion of the assignment, my hypothesis is that larger values of gamma will result in higher rewards, as the agent places more implication on actions that are further down the road than closer.

Because there were 36 linear combinations performed, I chose to include the best performing one for each gamma where epsilon = 0.1.



	Gamma	Epsilon	Time	Iterations	Reward	Max V	Mean V	Error	Success Rate
0	0.05	0.1	0.011192	2.0	1.052632	1.052632	0.780373	0.039568	0.166667
6	0.10	0.1	0.001001	2.0	1.111111	1.111111	0.814375	0.078238	0.833333
12	0.20	0.1	0.001018	2.0	1.250000	1.250000	0.896169	0.152643	0.666667
18	0.30	0.1	0.001004	2.0	1.428571	1.428571	1.003390	0.222547	0.666667
24	0.50	0.1	0.000996	2.0	2.000000	2.000000	1.360731	0.343770	0.500000
30	0.90	0.1	0.000999	3.0	10.000000	10.000000	7.636064	2.717639	0.333333

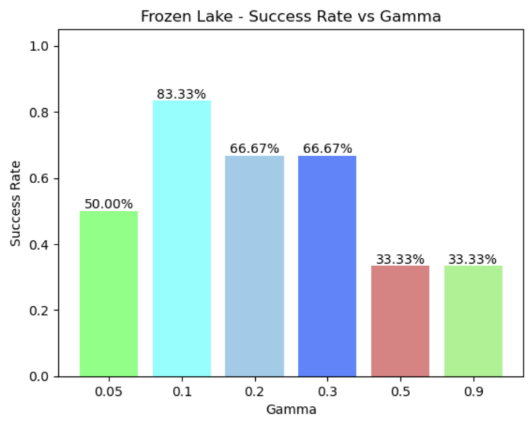
In general, policy iteration was much quicker than value iteration when it came to convergence:



Additionally, in comparison to value iteration, policy iteration places a heavier importance on later decisions and their implications:

The reason for earlier convergence, I think, is the fact that policy iteration is better at optimizing the exploration-exploitation problem: because our agent is able to first evaluate the current policy and then makes improvements. This two-step process can be more efficient in certain scenarios compared to the single-step process of value iteration.

Interestingly, I thought it should be noted that I tried several values of epsilon: [1e-1, 1e-2, 1e-3, 1e-5, 1e-8, 1e-12]. Checking to see which epsilon values led to higher rewards was fruitless: the max and mean score achieved by the agent was the same for each gamma when changing the epsilon values.



Additionally, it was interesting that as the values for policy iteration, both the max and average value, grew like a small exponential:

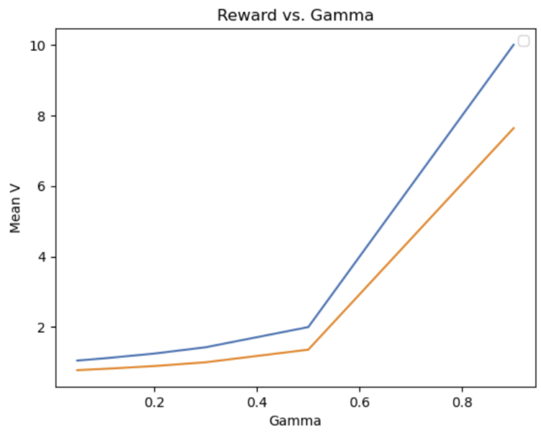


Fig. 1. blue - Max Value, orange - Mean Value

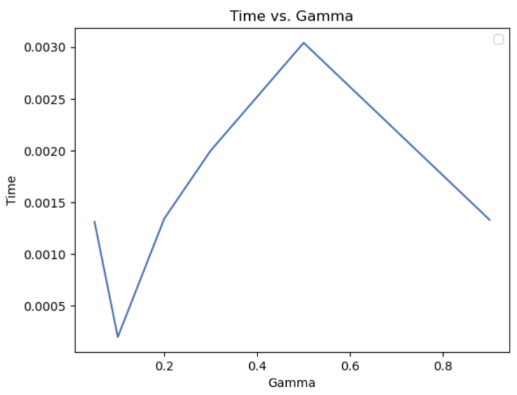
At a value of 0.5, we can see that the reward increases at a much quicker rate than before. This means that when we conduct policy iteration, our agent is placing much more emphasis on the consequences of their current actions, and the importance of later actions is much more consequential to it.

Also something of note is the fact that at gamma = 0.5, the computation time is the highest:

In the end, I was mostly correct in my hypothesis that a larger gamma value will result in higher rewards.

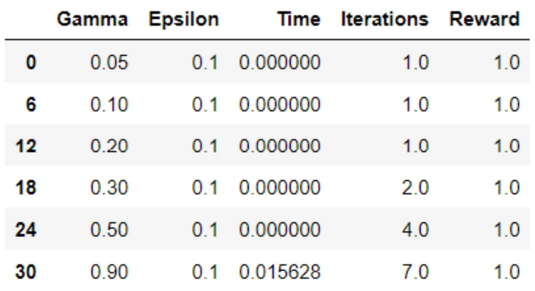
B. Value Iteration

In this section, we will explore how to iterate through ”values”. The value of a state is determined by iterating through a policy and finding the maximum action value for that state. Convergence of this problem is determined by the value of delta δ falling below a given threshold epsilon ϵ. Epsilon is the stopping criterion for the algorithms. I experimented with several different gamma γ and epsilon values. Gamma γ is the discount factor ranging from 0 to 1. It is used to determine the importance of future rewards in the decision making process. The larger the gamma value, the more the agent considers a long-term reward more heavily. The closer the value is to 0, the more the agent focuses on short term rewards. The algorithms will continue iterating until the change in the value function is smaller than epsilon.



Below, we can see the best values for each Gamma value. I experimented with 6 Gamma values and 6 Epsilon values. This would result in 36 linear combinations, and for the purposes of reporting, would be too clunky. Thus, I have picked to report a single best record for each Gamma. The way I determined this was by finding only those that converged (all 36 did), then finding the minimum number of iterations that took for that specific Gamma value.

My hypothesis is that again, the larger values for gamma will result in a higher reward.

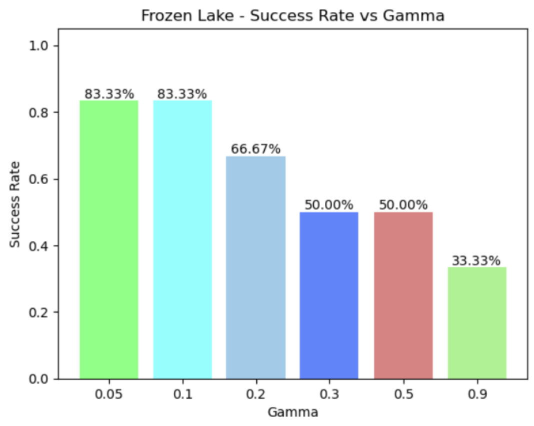


	Gamma	Epsilon	Time	Iterations	Reward
0	0.05	0.1	0.000000	1.0	1.0
6	0.10	0.1	0.000000	1.0	1.0
12	0.20	0.1	0.000000	1.0	1.0
18	0.30	0.1	0.000000	2.0	1.0
24	0.50	0.1	0.000000	4.0	1.0
30	0.90	0.1	0.015628	7.0	1.0

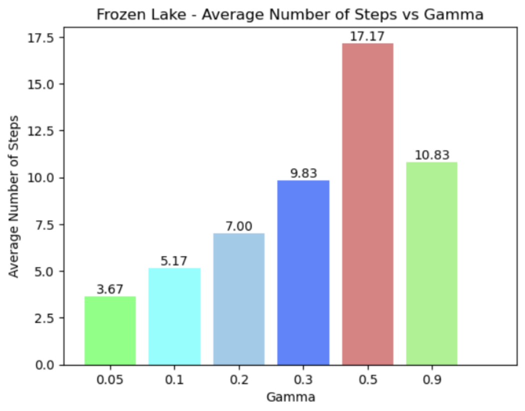
Interestingly, contrary to what I thought, the more risk-seeking seem to fare better (in the sense that they converge faster) than the more risk-averse agents. Also, the only agent that took some semblance of time was the one with a gamma value of 0.90, which is very risk-averse. What I takeaway from this is that in this context, being risk-averse is worse than just taking the risk.

Even more interestingly, the more risk-seeking gamma values fared better more often:

On average, not only did the lower Gammas converge faster, they actually were more successful than the more risk-averse gamma values. However, only two values did better than chance, those being the gamma values of 0.05 and 0.1. Very interestingly, the higher the values got for gamma, the worse the algorithm did on average.



This idea is also corroborated by the following graph that plots the average number of steps (until convergence) versus gamma.



The increase in average number of steps is pretty steady until we get to 0.5.

Overall, the agent is making decisions based on more immediate consequences of its actions. This short-sighted behavior could be risky when applied in actual contexts. However, for our purposes, this means that the frozen lake we generated had our elf able to move much quicker without thinking of possible ramifications later on.

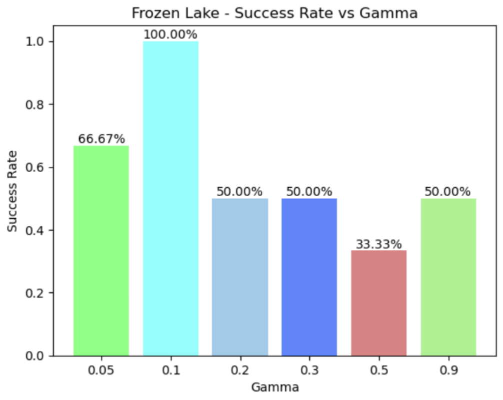
What I think is the most interesting thing here is that when I plotted my graph again for Time vs. Gamma with the value iteration, I got the exact same graph.

In the end, our agent (when using value iteration) preferred a lesser value of gamma than the agent when it came to policy iteration. This is evident via the success rate of the smaller values, and the decreasing success rate with an increase in gamma, our discount factor. This means that the agent would fare better doing things that did not involve long-term thinking. When smaller values of gamma are chosen, we are certain that our agent does not care for longer-term consequences.

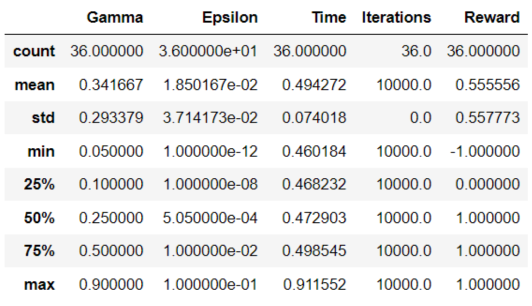
C. Q-learning

This section is a little different in the sense that as compared to policy and value iteration, there is no model here. This means the agent does not know what the transition probabilities are, nor does it know what the rewards are. In this section, we will explore how varying values of gamma, our discount factor from before, will fare in terms of success rate. An iteration with a specific gamma value is considered a success if the agent makes it across the lake and scores a 1. Additionally, alpha (α), the learning rate, will be held constant across all experiments in this section. We will hold α at 0.1. Ranging from 0-1, α determines how much new information is regarded and prior information is thrown away. A larger value of alpha will choose to accept and put more weight on newer information, whereas a smaller alpha will choose to prioritize older information. My hypothesis for this portion is that middle values of gamma will fare better. When the agent can make decisions off of both current and prior information, without completely disregarding one or the other, they will do better on average (I think).

This hypothesis was actually correct: after iterating each gamma value 10000 times upon convergence, the middle values did very well:

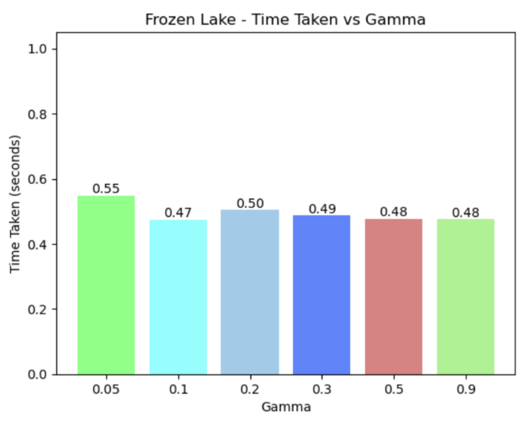


Overall, the agent performed much better than with policy and value iteration. Each gamma, excluding 0.5, performed better than prior:



	Gamma	Epsilon	Time	Iterations	Reward
count	36.000000	3.600000e+01	36.000000	36.0	36.000000
mean	0.341667	1.850167e-02	0.494272	10000.0	0.555556
std	0.293379	3.714173e-02	0.074018	0.0	0.557773
min	0.050000	1.000000e-12	0.460184	10000.0	-1.000000
25%	0.100000	1.000000e-08	0.468232	10000.0	0.000000
50%	0.250000	5.050000e-04	0.472903	10000.0	1.000000
75%	0.500000	1.000000e-02	0.498545	10000.0	1.000000
max	0.900000	1.000000e-01	0.911552	10000.0	1.000000

In our pursuit of the exploration-exploitation tradeoff, I wanted to spend time (no pun intended) discussing how much time is taken for one iteration to converge of each specific gamma value. Below is the graph that demonstrates the average amount of taken. Each gamma value had the average amount of time taken over the course of 10000 iterations.



In the end, over the long-term, we can see that the time averages to about 0.50 seconds per iteration, regardless of the gamma value being used. That being the case, it shows us that for the Frozen Lake problem that we have set up (it may be different under different circumstances), our agent does not really care about the discount factor. This, intuitively almost makes sense to me. In q-learning, our agent knows nothing about the transition probabilities, nor the rewards. They simple iterate through and try to find the best solution. Alpha, the learning rate (which we will explore more of in the q-learning section of Jack's Car Rental), likely plays a bigger role: when all the agent has to really go off of is whether to value current information more or prior information more, choosing that becomes much more vital in solving the problem.

Overall, my hypothesis that middle values of gamma will perform better kind of ended up not really making sense in terms of this problem solving method. As explained above, the learning rate matters more.

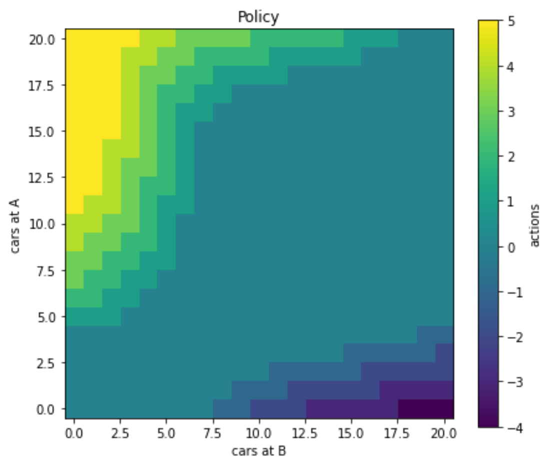
III. MDP 2: Jack's Car Rental Problem

In the first problem that we solved, we looked at a grid-world problem. This means we had a matrix-like structure to the Markov Decision Problem. This is not the case for Jack's Car Rental. The problem is as follows (taken directly from Towards Data Science): "Jack manages two locations for a nationwide car rental company. Each day, some number of customers arrive at each location to rent cars. If Jack has a car available, he rents it out and is credited $10 by the national company. If he is out of cars at that location, then the business is lost. Cars become available for rent the day after they are returned. To help ensure that cars are available where they are needed, Jack can move them between the two locations overnight, at a cost of $2 per car moved. We assume that the number of cars requested and returned at each location is a Poisson random variable".

What does it mean for this problem to be solved? In this context, it is two-fold. First, and likely more obviously, Jack is trying to maximize profit. Second, once Jack find's this strategy (policy) that maximizes profit, how can he compare which situations are better than others? This is what we will be solving here on out.

The graphs in these next three sections will look very different than the ones for the Frozen Lake problem. The reason for this is that the objective of Frozen Lake was simple and constant: make it to the other side of the lake without falling into a hole. Seems easy enough. However, there is no constant value in the car rental problem. I can assign a +1 to the agent as a reward for making to the other side of the lake. This does not make sense to do for the car rental problem, as we are trying to maximize profit, and there is no cap on that.

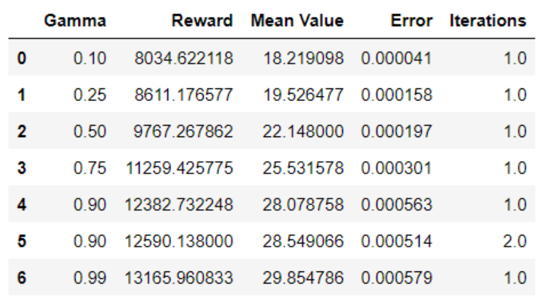
The picture below is what a sample policy/value iteration heat-map may look like for Jack's Car Rental:



The y-axis on the left represents cars at Rental Car Location A, and the x-axis on the bottom represents cars at Rental Car Location B. The value on the heat map represents the profit earned at that specific combination.

A. Policy Iteration

In this section, I wanted to explore policy iteration and how different gamma values resulted in different rewards.



	Gamma	Reward	Mean Value	Error	Iterations
0	0.10	8034.622118	18.219098	0.000041	1.0
1	0.25	8611.176577	19.526477	0.000158	1.0
2	0.50	9767.267862	22.148000	0.000197	1.0
3	0.75	11259.425775	25.531578	0.000301	1.0
4	0.90	12382.732248	28.078758	0.000563	1.0
5	0.90	12590.138000	28.549066	0.000514	2.0
6	0.99	13165.960833	29.854786	0.000579	1.0

First, the biggest difference I noticed was simply the iterations: when comparing the number of iterations, even on average, policy iteration outperforms value iteration. The average number of iterations for policy iteration was 2.25, whereas for value iteration (see below), was 11.75. This is almost 5 times worse.

The following graphs illustrate the efficacy of each policy with their respective gammas applied:

From my experiments and the graphs above, the agent (Jack in this case) would fare best by spending his efforts thinking

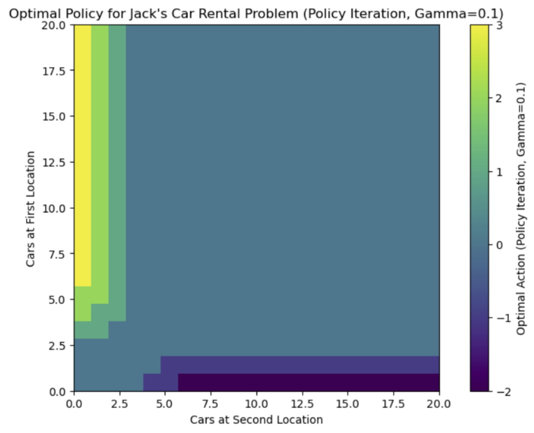


Fig. 2. Gamma = 0.1

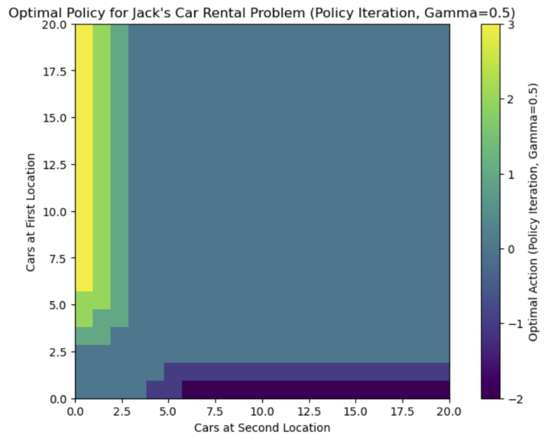


Fig. 3. Gamma = 0.50

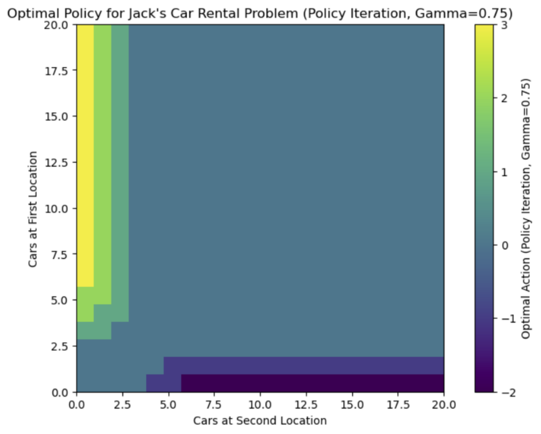


Fig. 4. Gamma = 0.75

about the consequences of his actions long term. However, once he reaches a certain point, there is minimal value gained for the much larger sustained efforts he would likely have to put into that much planning. Interestingly, there seems to be a diminishing margin of return happening here: as the values

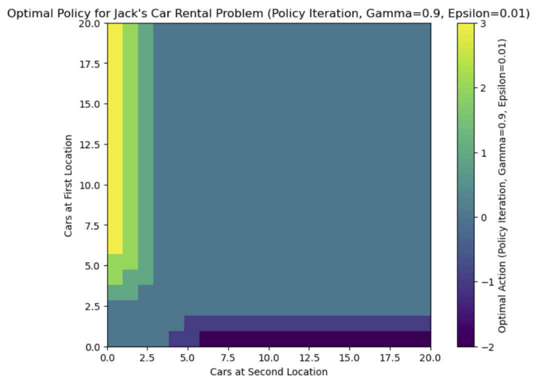


Fig. 5. Gamma = 0.9

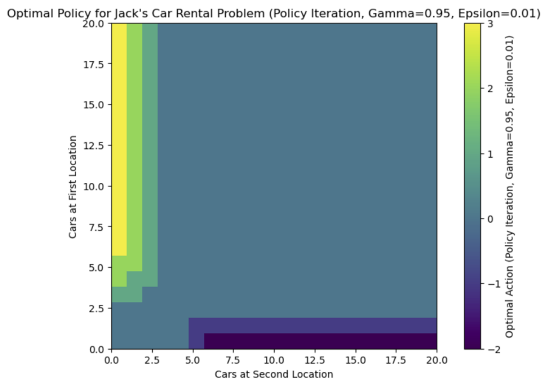


Fig. 6. Gamma = 0.95

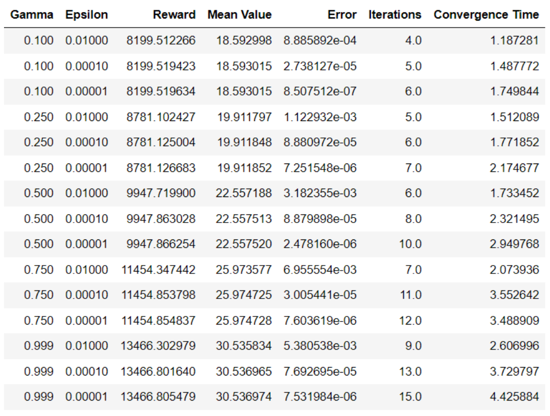
of gamma push into the 0.70s and onward, there is not much of a value increase.

In terms of wall clock time, this was the fastest of the three iterations for Jack's Car Rentals: CPU times: total: 8.5 s, Wall time: 11.9 s. I think this is because policy iteration does a better job than value iteration in the exploration-exploitation dilemma, focusing on one thing at time, as opposed to value iteration. This can optimize the process. Additionally, because epsilon does not play a factor here, there is one less loop for iterating through, bringing the time down by a factor of n.

In the end, we can see that as our gamma value increases, it becomes increasingly important that the number of cars at Rental Lot B lessens. However, there are many iterations with gamma that show that it does not make a difference from gamma = 0.75 to gamma = 0.9 if Jack changes his strategy. This shows us that Jack is better off with a mixed strategy - think somewhat in the moment, and somewhat for the future if he wants to maximize his profit.

B. Value Iteration

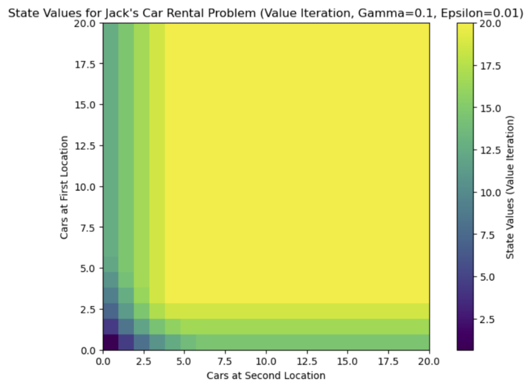
In this section, we explore the effects of the discount rate - gamma γ on our reward. Reward in this case the number of cars at a lot A. Below, we can see our results. Experimenting with 5 different values for gamma, and 3 different values for epsilon, we can see the plots below for overall performance.

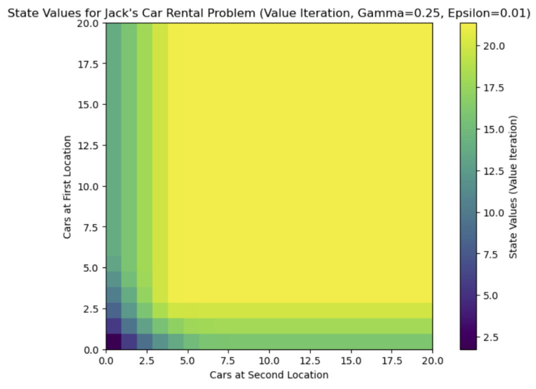


Gamma	Epsilon	Reward	Mean Value	Error	Iterations	Convergence Time
0.100	0.01000	8199.512266	18.592998	8.885892e-04	4.0	1.187281
0.100	0.00010	8199.519423	18.593015	2.738127e-05	5.0	1.487772
0.100	0.00001	8199.519634	18.593015	8.507512e-07	6.0	1.749844
0.250	0.01000	8781.102427	19.911797	1.122932e-03	5.0	1.512089
0.250	0.00010	8781.125004	19.911848	8.880972e-05	6.0	1.771852
0.250	0.00001	8781.126683	19.911852	7.251548e-06	7.0	2.174677
0.500	0.01000	9947.719900	22.557188	3.182355e-03	6.0	1.733452
0.500	0.00010	9947.863028	22.557513	8.879898e-05	8.0	2.321495
0.500	0.00001	9947.866254	22.557520	2.478160e-06	10.0	2.949768
0.750	0.01000	11454.347442	25.973577	6.955554e-03	7.0	2.073936
0.750	0.00010	11454.853798	25.974725	3.005441e-05	11.0	3.552642
0.750	0.00001	11454.854837	25.974728	7.603619e-06	12.0	3.488909
0.999	0.01000	13466.302979	30.535834	5.380538e-03	9.0	2.606996
0.999	0.00010	13466.801640	30.536965	7.692695e-05	13.0	3.729797
0.999	0.00001	13466.805479	30.536974	7.531984e-06	15.0	4.425884

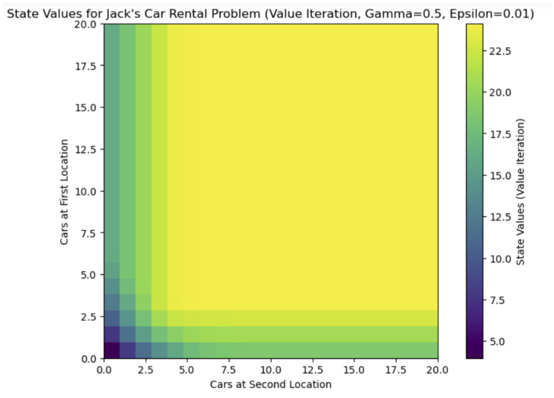
With no surprise at all, there is clearly a positive correlation between the gamma value and number of iterations it takes to converge. When gamma was equal to 0.1, the number of iterations was the lowest. As gamma went up to 0.75 and even 0.90, the number of iterations doubles, if not triples.

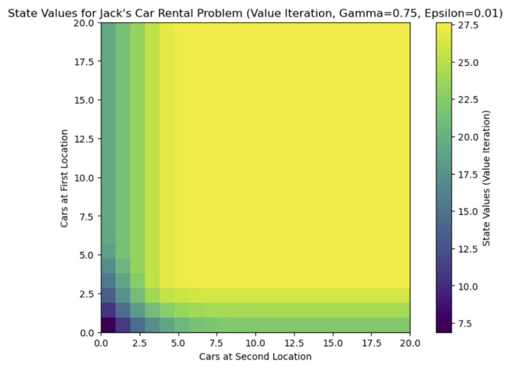
For simplicity and ease of comparison, I chose to hold epsilon ϵ constant at 0.01, while we explore the differences when gamma changes. See below for the graphs:

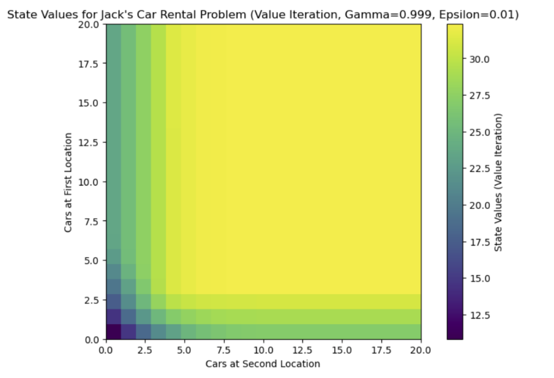




Very interestingly, we do not see much of a change in the values whatsoever when we plot these graphs: whether Jack chooses to think and act on current values, or thinks about later consequences of his current actions, the value does not change too much.





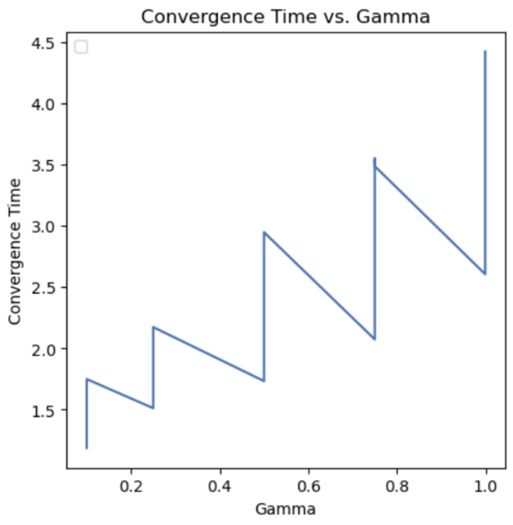


Here, I think this can be attributed to a low epsilon value. Because the error threshold is a larger value, convergence is met much faster than smaller values of epsilon. This is consequential in choosing the right move, because any small change in the value function might cause it to converge in the context of such a large threshold.

Next, in the interest of the exploration-exploitation tradeoff, I explored the convergence time and how it was effected by our choice in gamma.

We get this very interesting looking shape, which can be attributed to the epsilon values. There is very clearly a positive correlation between convergence time and gamma, but this spiky, non-linear increase must be due to the varying epsilon values. As the stopping criteria change with respect to gamma, so too does the convergence time.

Lastly, although value iteration actually has the most pa-



rameters that can vary, actually computed very quickly for all 15 of these linear combinations: CPU times: total: 35.3 s, Wall time: 42.6 s. This was surprising to me, as I expected this to be just a tad-bit quicker than q-learning.

C. Q-learning

In this section, I will explore the relationship between alpha and rewards. Alpha α is used as the learning, and it determines the extent to which the newly acquired information should overwrite the existing information in the Q-values. It influences the weight given to the most recent experiences when updating the Q-values. I hypothesize that lesser values of alpha α will perform better, because they will not just take the current value and run with it.

Of all 6 solutions I attempted for both problems, q-learning for this problem took by far the longest amount of time to complete. (I later realized that epsilon does not actually factor in here. My brain was burnt out. Please read the conclusions and final takeaways section on how this affected the work I did for this assignment).

The times were: CPU times: total: 11h 39min 43s, Wall time: 22h 56min 37s.

Everything else being constant (gamma γ = 0.9), I varied the learning rate (α). The results were as follows:

Additionally, q-learning performed the worst of all:

In this graph, we can see that the Mean Value at each Iteration progressively gets worse. This shows that Jack is likely not relying the learned q-values, but rather random actions. To further corroborate this, we can look at the summary statistics provided by the DataFrame below:

The average value achieved by q-learning was -56, which is staggeringly different from the policy iteration we did above. Because there was no access to transition probabilities nor rewards, and Jack chose not to learn from the q-values, it could be the case his random decisions just kept leading to poorer and poorer outcomes.

Lastly, we explore the exploration-exploitation trade-off with an epsilon of 1e-5, and a decay rate of 0.0001.

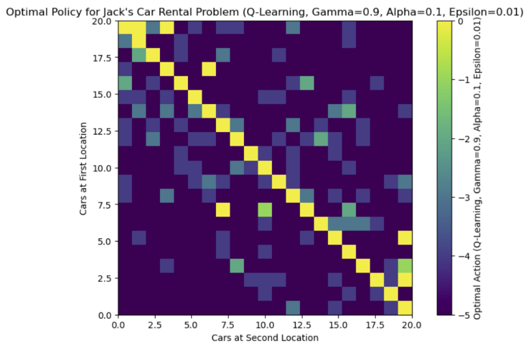


Fig. 7. alpha = 0.1

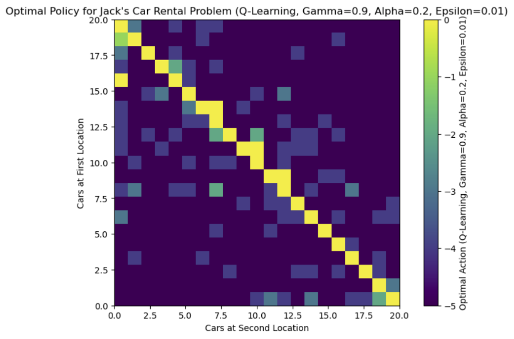


Fig. 8. alpha = 0.2

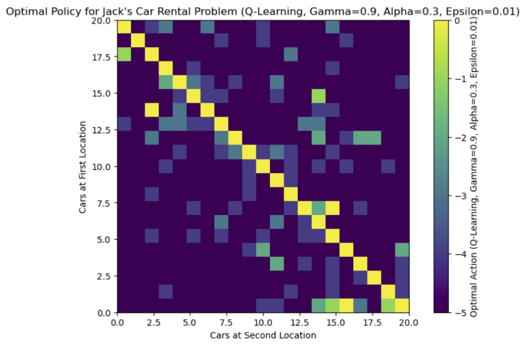


Fig. 9. alpha = 0.3

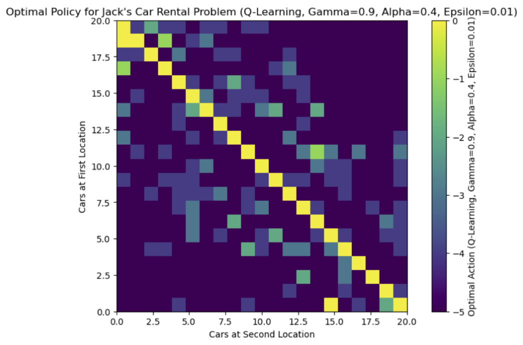
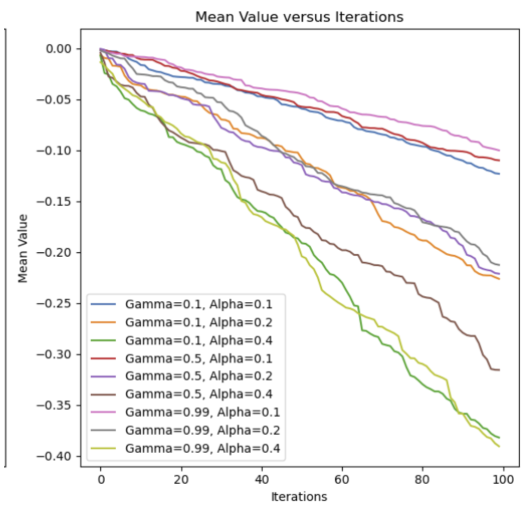
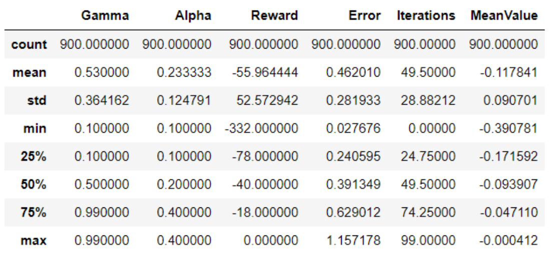


Fig. 10. alpha = 0.4





	Gamma	Alpha	Reward	Error	Iterations	MeanValue
count	900.000000	900.000000	900.000000	900.000000	900.00000	900.000000
mean	0.530000	0.233333	-55.964444	0.462010	49.50000	-0.117841
std	0.364162	0.124791	52.572942	0.281933	28.88212	0.090701
min	0.100000	0.100000	-332.000000	0.027676	0.00000	-0.390781
25%	0.100000	0.100000	-78.000000	0.240595	24.75000	-0.171592
50%	0.500000	0.200000	-40.000000	0.391349	49.50000	-0.093907
75%	0.990000	0.400000	-18.000000	0.629012	74.25000	-0.047110
max	0.990000	0.400000	0.000000	1.157178	99.00000	-0.000412

Below, we can see an example of the exploration-exploitation tradeoff from Towards Data Science.

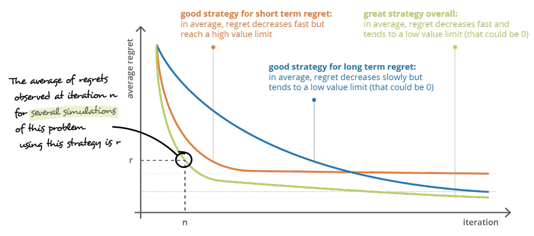
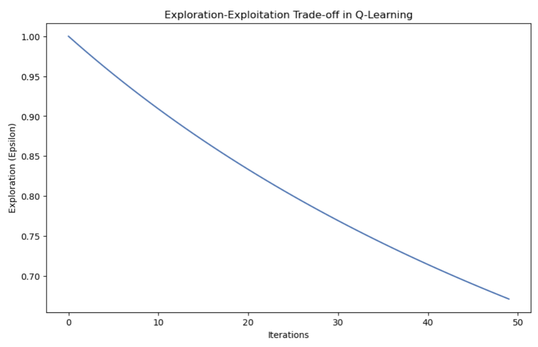


Fig. 11. Example Exploration vs. Exploitation Graph from Towards Data Science

Below is a graph of our exploration-exploitation trade-off.



From this graph, we can determine that if our agent (Jack) sticks with this plan, it is a good course of action for future regret. Why is this the case? We can clearly see that our agent is gradually relying on the learned q-values, not just random action, as is the case in the early stages. Why is this the case for earlier stages? Because q-learning is model free. Jack knows nothing about the transition probabilities nor rewards. He can later rely on the learned q-values, which is not a possibility early on.

In the end, my initial hypothesis that smaller alpha values will perform better was indeed correct, but not by much, as q-learning performed poorly overall.

IV. Conclusions & Final Takeaways

I realized later that epsilon does not actually have any role in policy iteration and q-learning, but because of computation's sake, I had to leave it in the dataframe. For policy iteration and q-learning, please disregard the epsilon values.

What I took away from making this mistake is the sheer cost of time complexity: adding another for loop increases my big O by n times. I could have saved a lot of time. Q-learning already is resource-consuming because of its inherent nature. By adding an extra loop, I was now spending O(n^3) instead of just O(n^2) time.

Even with a relatively small grid world problem, and a non-grid world problem like Jack's Car Rental, it can be extremely strenuous to try and compute with a mistake like mine. When problems become more complex and closer to real-world-like, I can only imagine how much more expensive a computational error like mine would cost, both computationally and physically.

Lastly, please excuse the formatting. For some reason, Overleaf does not like to keep text with pictures, despite me having it written in the correct places. Please use the captions and titles for each image if necessary.